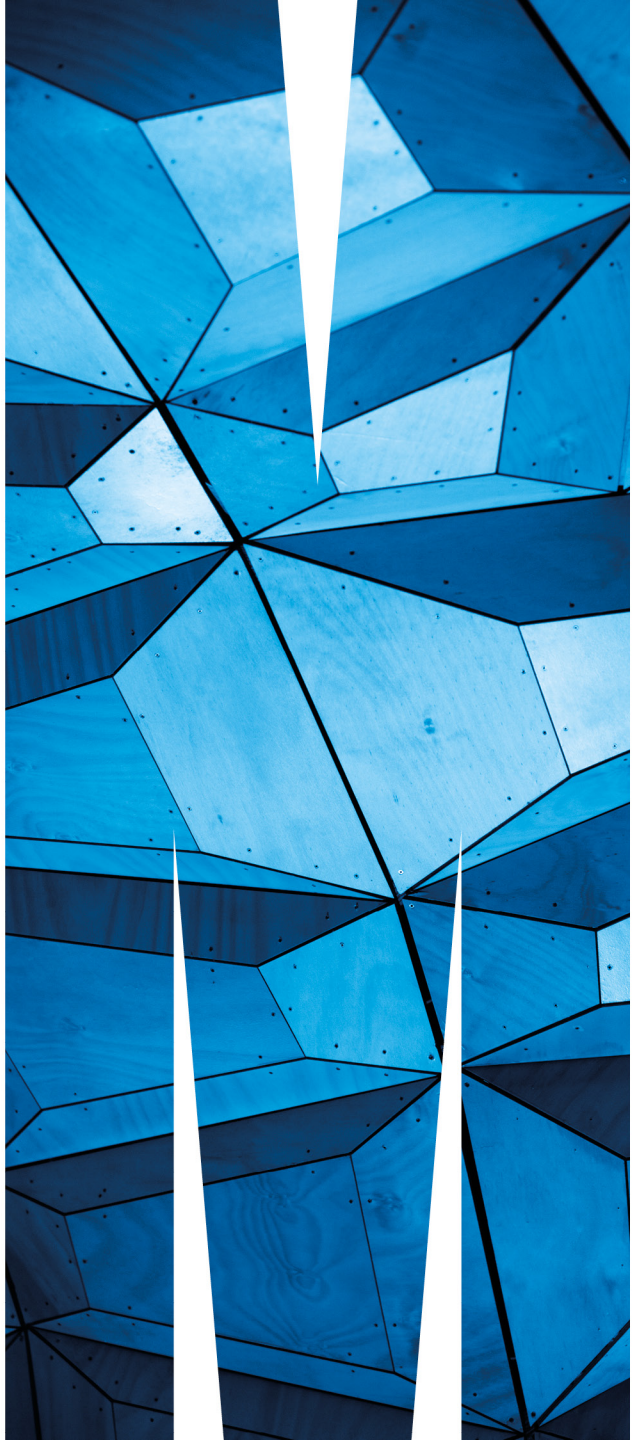
MONASH University

Earliest Deadline First Scheduling

Clive Maynard



Earliest Deadline First (EDF) scheduling

EDF is a dynamic priority scheduling scheme (unlike RMA)

The process closest to its deadline has the highest priority

This requires recalculating the state of processes at every timer interrupt.

For analysis it is necessary to ensure that every process meets its deadlines for all scheduling possibilities.

This means we need to evaluate all time periods up to the least common multiple (LCM) of the task periods.

Earliest Deadline First (EDF) scheduling

EDF is an *optimal* scheduling algorithm on preemptive uniprocessors, in the following sense: if a collection of independent *tasks,* each characterized by an arrival time, an execution requirement and a deadline, can be scheduled (by any algorithm) in a way that ensures all the jobs complete by their deadline, the **EDF** will schedule this collection of jobs so they all complete by their deadline.
With scheduling periodic processes that have deadlines equal to their periods, **EDF** has a utilization bound of 100%. Thus, the schedulability test for **EDF** is:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1,$$

where the{Ci} are the worst-case computation-times of the n processes and the {Ti} are their respective inter-arrival periods (assumed to be equal to the relative deadlines).
That is, EDF can guarantee that all deadlines are met provided that the total CPU utilization is not more than 100%. Compared to fixed priority scheduling techniques like rate-monotonic scheduling, **EDF** can guarantee all the deadlines in the system at higher loading.

Earliest Deadline First (EDF) scheduling

When the system is overloaded, the set of processes that will miss deadlines is largely unpredictable (it will be a function of the exact deadlines and time at which the overload occurs.) This is a considerable disadvantage to a real time systems designer.
The algorithm is also difficult to implement in hardware and there is a tricky issue of representing deadlines in different ranges (deadlines can not be more precise than the granularity of the clock used for the scheduling).
If modular arithmetic is used to calculate future deadlines relative to now, the field storing a future relative deadline must accommodate at least the value of the (("duration" {of the longest expected time to completion} * 2) + "now").
Therefore **EDF** is not commonly found in industrial real-time computer systems.

Instead, most real-time computer systems use fixed priority scheduling (usually rate-monotonic scheduling). With fixed priorities, it is easy to predict that overload conditions will cause the low-priority processes to miss deadlines, while the highest-priority process will still meet its deadline.

EDF example

To distinguish processes/tasks from processors this is using T1,T2 and T3 and P0

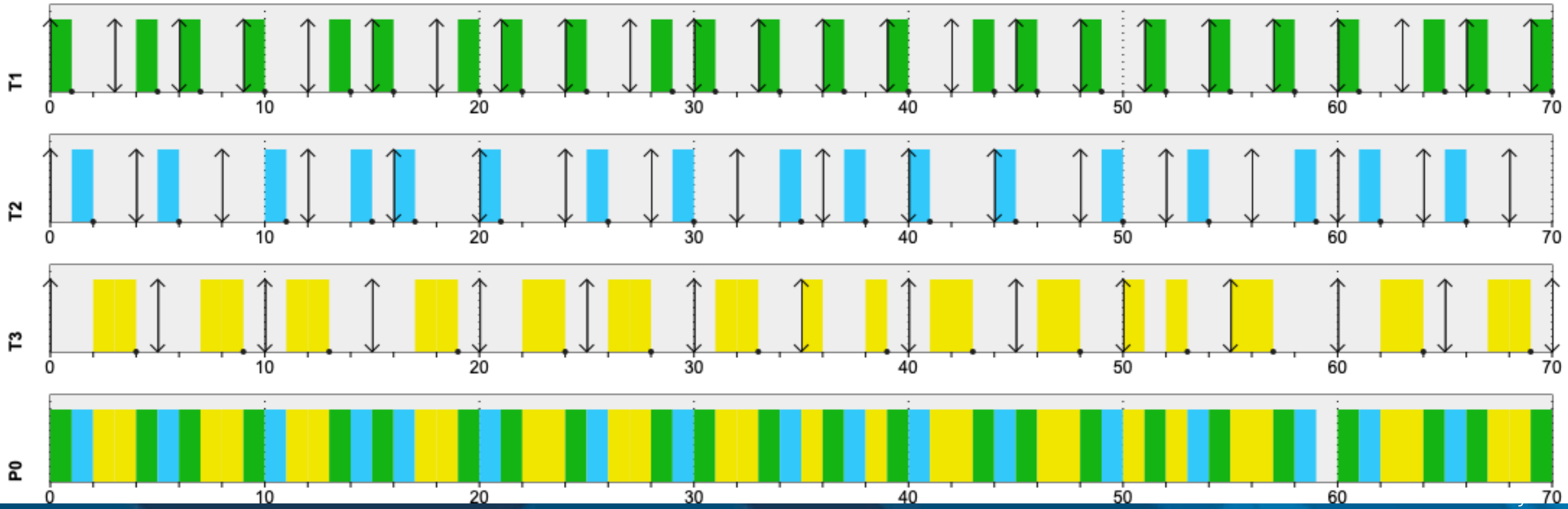
Process	Execution time	Period
T1	1	3
T2	1	4
T3	2	5

Least common multiple (LCM) of periods is 60.

Utilization is 1/3+1/4+2/5 = 0.9833

EDF Example

Process	Execution time	Period
T1	1	3
T2	1	4
T3	2	5



Time	Running Process	Deadlines (end of time)
0		
1		
2		P1
3		P2
4		P3
5		P1
6		
7		P2
8		P1
9		P3
10		
11		P1 P2
12		
13		
14		P1 P3
15		P2

Tabular analysis of the EDF process

Enter the deadlines for each of the tasks.
Note: Taken from the text with P1, P2 and P3

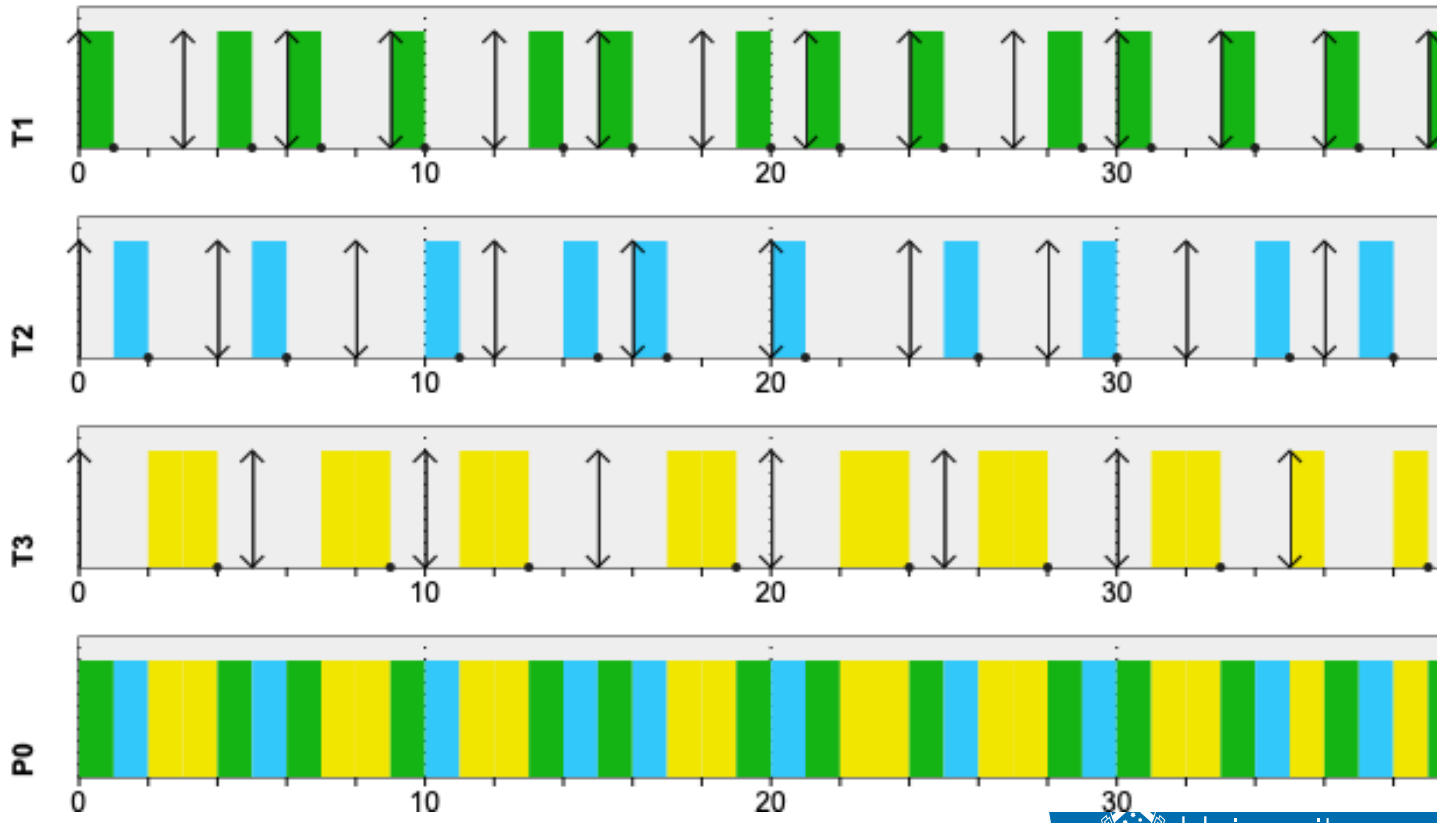
P1 has a period of 3 so it has to complete before completion of Time 0, 1 and 2

This tabular layout extends until the LCM of the periods

Time	Running Process	Deadlines (end of time)
0	P1	
1	P2	
2	P3	P1
3	P3	P2
4	P1	P3
5		P1
6		
7		P2
8		P1
9		P3
10		
11		P1 P2
12		
13		
14		P1 P3
15		P2

Don't forget

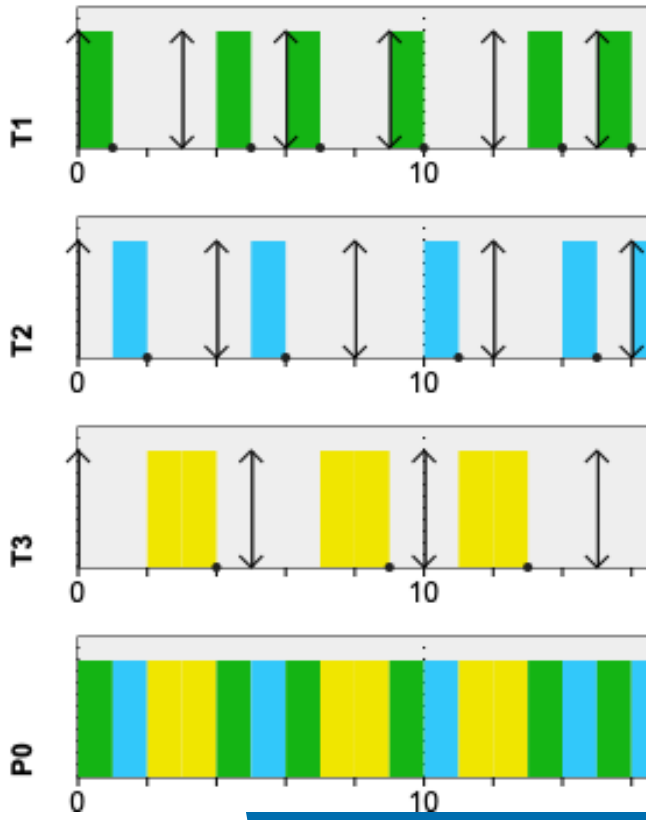
Table P1 is equivalent to the Gantt chart T1!!!!!



Time	Running Process	Deadlines (end of time)
0	P1	
1	P2	
2	P3	P1
3	P3	P2
4	P1	P3
5	P2	P1
6	P1	
7	P3	P2
8	P3	P1
9	P1	P3
10	P2	
11	P3	P1 P2
12	P3	
13		
14		P1 P3
15		P2

Note:
P1 and P2 are not scheduled until the end of time period 11 so P3 can execute for this period

At the end of time period 11 the situation is that P1 and P3 have the same time (3) to their respective deadlines.
There is a need to invoke an extra scheduling rule:
Given a choice between scheduling two tasks choose the one which is already executing



MONASH University

Results

Start date (ms): 0 End date (ms): 70

'Results'

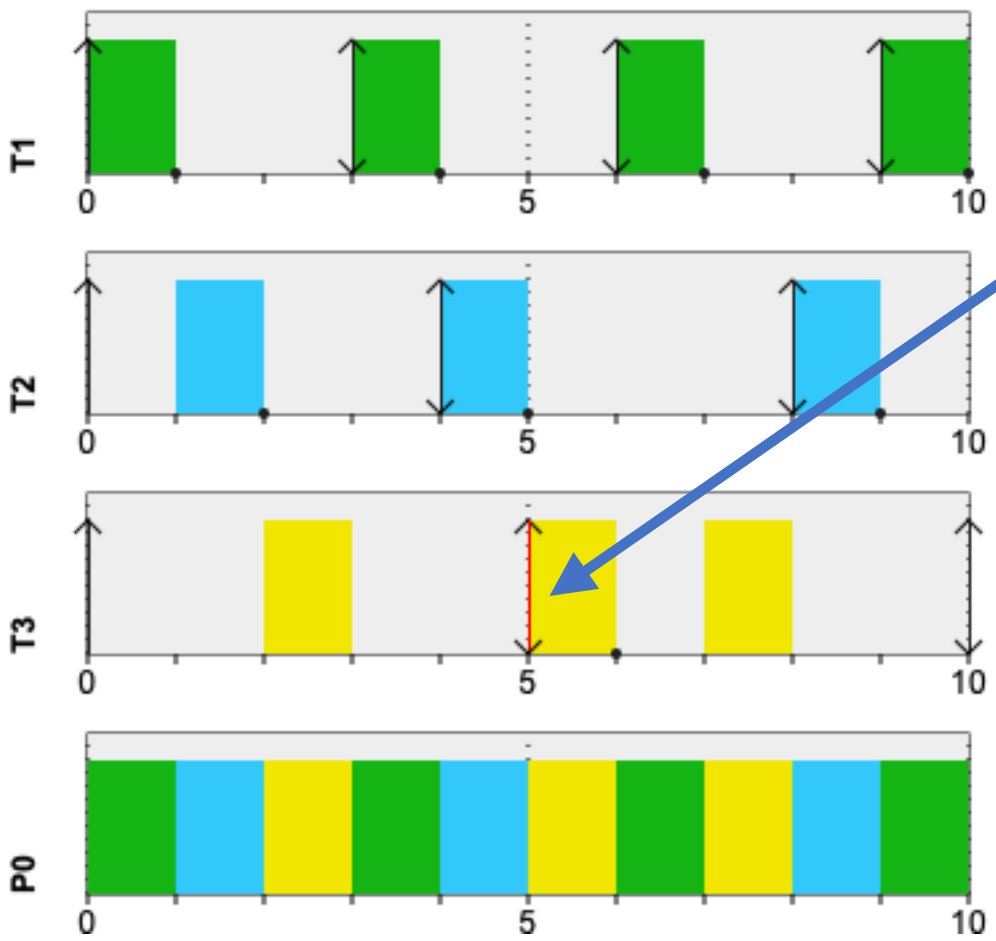
Gantt | General | Tasks | Processors | Logs | Scheduler

Date (cycles)	Date (ms)	Message
0	0	T1_1 Activated.
0	0	T2_1 Activated.
0	0	T3_1 Activated.
0	0	T1_1 Executing on P0
1000000	1	T1_1 Terminated.
1000000	1	T2_1 Executing on P0
2000000	2	T2_1 Terminated.
2000000	2	T3_1 Executing on P0
3000000	3	T1_2 Activated.
3000000	3	T3_1 Preempted! ret: 10000
3000000	3	T3_1 Executing on P0
4000000	4	T2_2 Activated.
4000000	4	T3_1 Terminated.
4000000	4	T1_2 Executing on P0

SimSo Logfile output

MONASH University

RMA with the EDF example



T3 misses its deadline with RMA scheduling

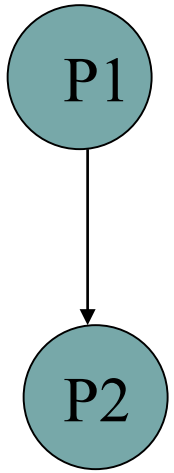
Fixing scheduling problems

- What if your set of processes is unschedulable?
  - Change deadlines in requirements.
  - Reduce execution times of processes.
  - Get a faster CPU.
  - Employ an accelerator.

Data dependencies

Data dependencies allow us to improve utilisation.

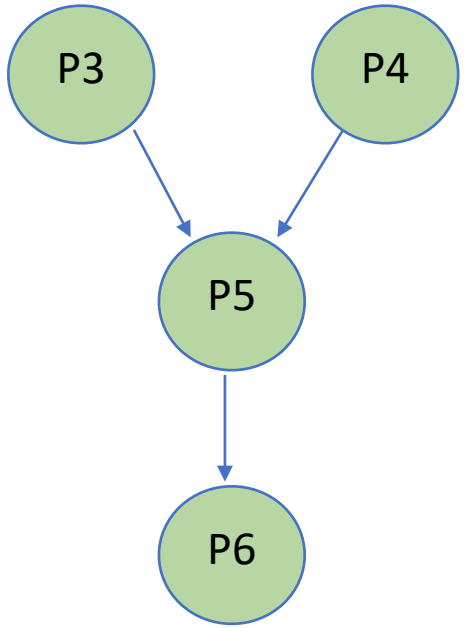
P1 and P2 cannot run simultaneously



P3 and P4 can run simultaneously

P5 needs to wait until both P3 and P4 have completed.

P6 is dependent on P5



Least Laxity first (LLF) Scheduling

Least Laxity First (LLF) is a task level dynamic priority scheduling algorithm.
It means that every instant is a scheduling event because laxity of each task changes on every instant of time.
A task which has least laxity at an instant, it will have higher priority than others at this instant.

Investigate LLF as a possible scheduling algorithm.
What happens when more than one task has the same laxity?

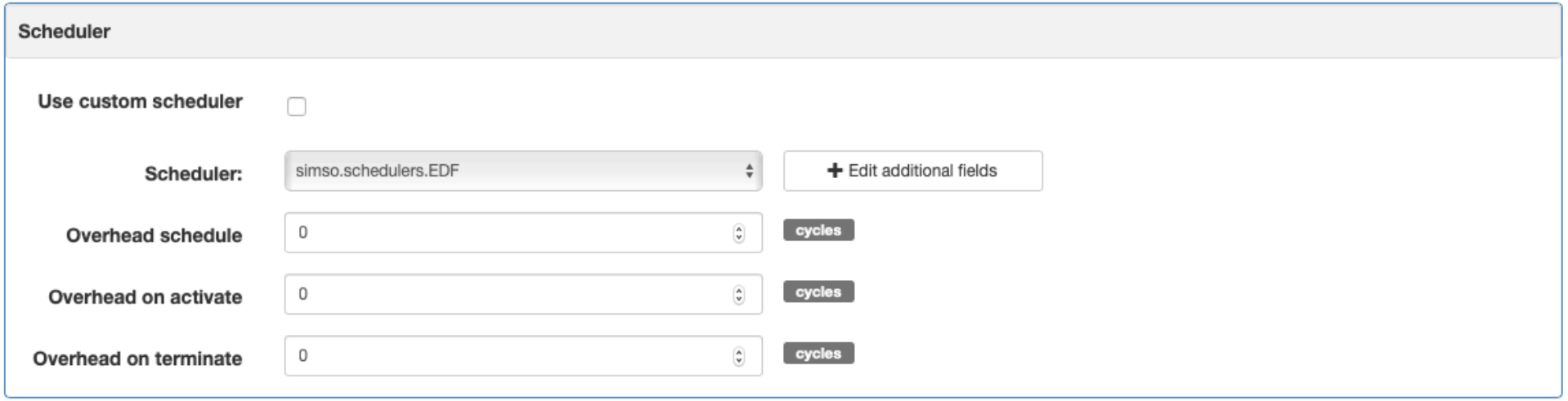
Context-switching time

Non-zero context switch time can push limits of a tight schedule.

Hard to calculate effects---depends on order of context switches.

In practice, OS context switch overhead is generally small.

In SimSo you can configure context switching overhead



Finite Context Switching Time

A "Rule of Thumb"

In most of our simple analyses we assume the context switching time is negligible but this may not always be true.

Engineers are pragmatic so it is common to have an estimation process to determine if we have a problem.

Rule of Thumb:

Evaluate the available idle time (T) up to the critical moment in the schedule (at the end of the first idle time) assuming negligible context switching time.

Determine the number of scheduling points (S) up to that time.

The worst case (maximum) context switching time should be less than T/(2S).

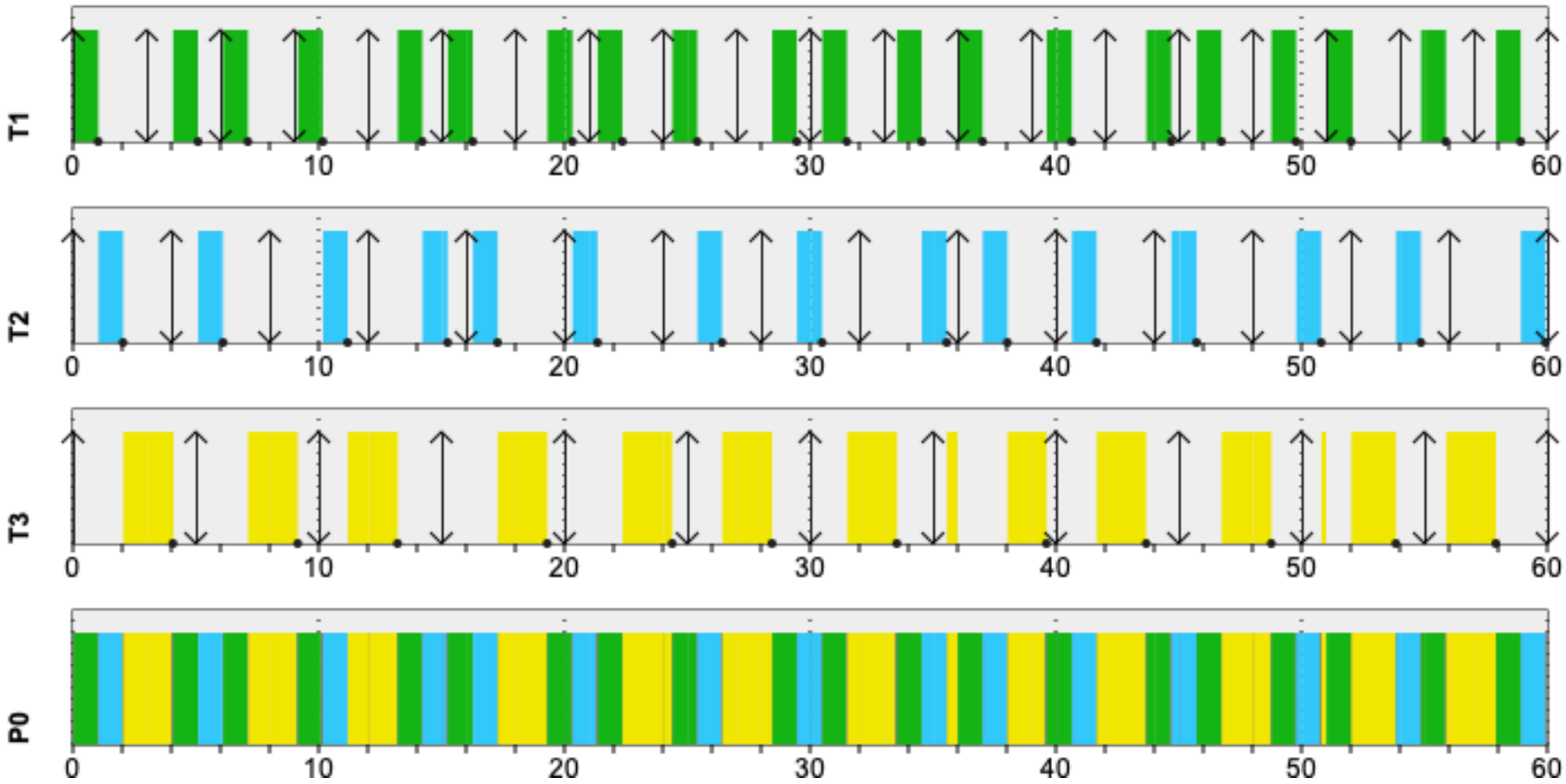
EDF Example

47 scheduling points and 1ms idle time upto the critical instant.

Max context switch time using

the rule of thumb

= 0.01



Look at the log file to see when execution was complete

Outside the Box

How does a context switch work?

When execution enters a context switch it needs to perform the following actions:

- Determine the highest priority ready task.
- IF not the current task
  - save the context of the current task
  - restore the context of the new task
- Transfer execution to the new task.

The user wants short execution time and predictable maximum execution time preferably independent of the priority of the new task.

Two scenarios:

- Same task: get back to it as soon as possible
- Different task: predictable (and preferably short constant) execution time.

Finding the highest priority ready task.

This depends highly on the way in which the information is stored.

How does uCOSII do it?

The ready list consists of two variables:

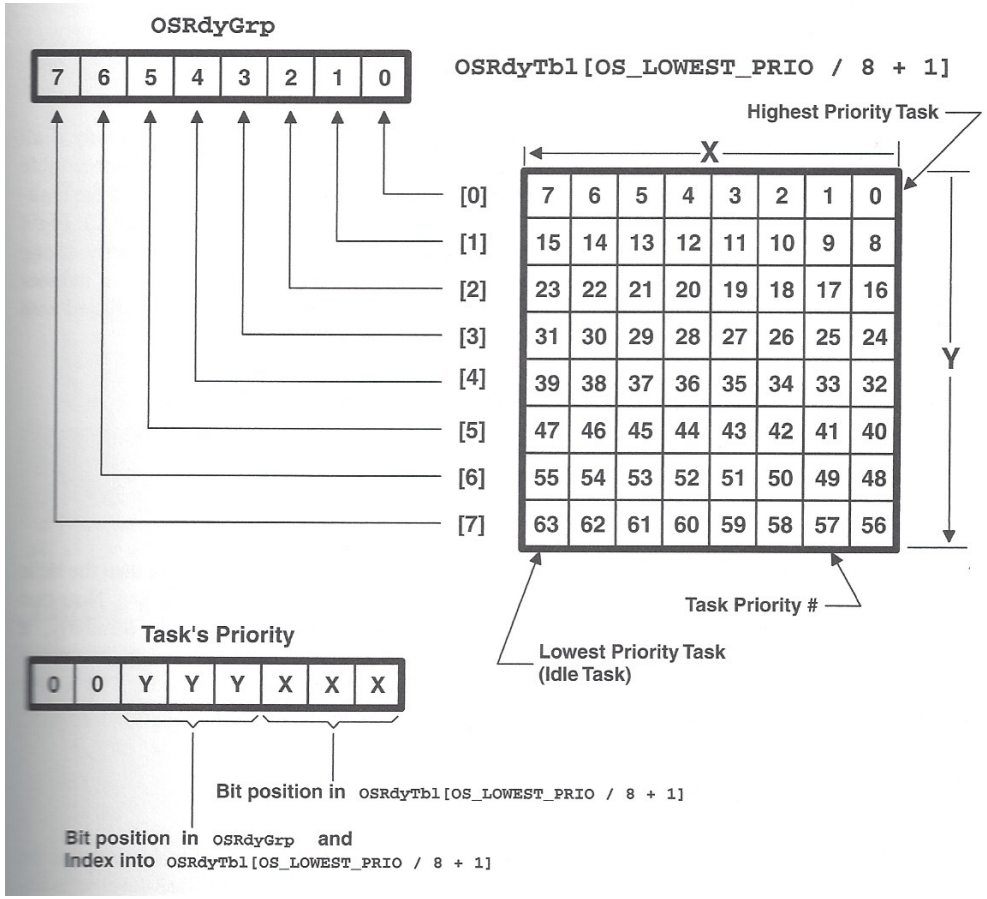
OSRdyGrp and OSRdyTbl[]

Task priorities are grouped(eight tasks per group) in OSRdyGrp. Each bit indicates when a task group is ready to run.

When a task is ready to run it sets the corresponding bit in OSRdyTbl[]

Example: If the task with priority 20 is ready to run then

Bit 2 is set in OSRdyGrp and there is one bit set in OSRdyTbl[2] in this case bit 4



Organising for minimal execution.

A lookup table is much faster than a search!!!

OSMapTbl[] - created to help deal with recording that a task is ready to run

Index	Bit Mask(binary)
0	00000001
1	00000010
2	00000100
3	00001000
4	00010000
5	00100000
6	01000000
7	10000000

Making a task ready to run.

OSRdyGrp |= OSMapTbl[prio >> 3];

OSRdyTbl[prio >> 3] |= OSMapTbl[prio & 0x07];

****

Example: prio of 20 or 0b00010100

prio >> 3 is 0b00000010 or 0x02

OSMapTbl[2] is 0b00000100 or 0x04

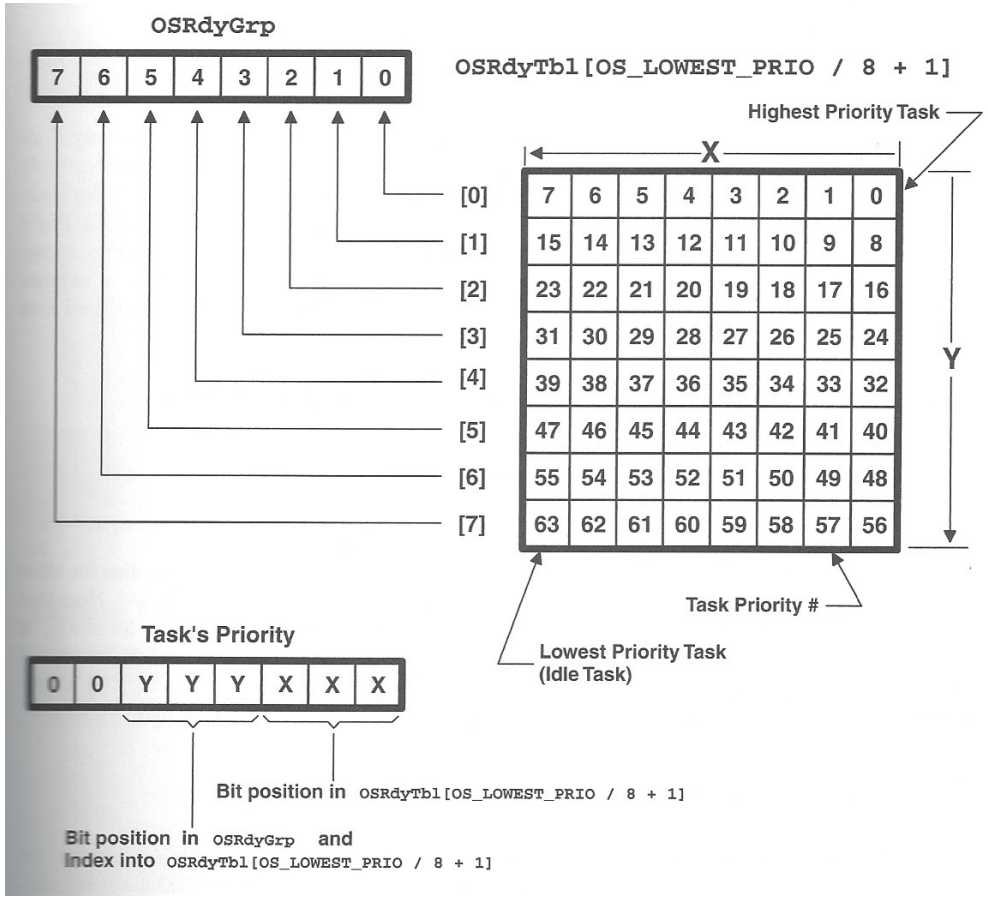
So we are ORing into OSRdyGrp this value

and for the second part

prio & 0x07 = 0x04 so

OSMapTbl[4] is 0b00010000 or 0x10

OSRdyTbl[2] is ORd with that value to put a 1 in the bit corresponding to priority 20.



Finding the highest priority task ready to run

Investigate this process by searching uCOSII documentation.

It has similarities to the table lookup process just described.

What is the minimum number of 1s to be found in the table? Why?